Preprint
UCRL-JC-142172

Are Multiple Runs Better Than One?

E. Cantú-Paz

This article was submitted to
Genetic and Evolutionary Computation Conference 2001, San Francisco, CA, July 7-11, 2001

January 4, 2001

Are Multiple Runs Better than One?

Erick Cantú-Paz
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
7000 East Avenue, Livermore, CA 94550
cantupaz@llnl.gov

Abstract

This paper investigates whether it is better to use a certain constant amount of computational resources in a single run with a large population, or in multiple runs with smaller populations. The paper presents the primary tradeoffs involved in this problem and identifies the conditions under which there is an advantage to use multiple small runs. The paper uses an existing model that relates the quality of the solutions reached by a GA with its population size. The results suggest that in most cases a single run with the largest population possible reaches a better solution than multiple isolated runs. The findings are validated with experiments on functions of varying difficulty.

1 INTRODUCTION

Suppose that we are given a fixed amount of function evaluations to solve a particular problem with a genetic algorithm (GA). How would we divide these evaluations to maximize the expected quality of the solution? One possibility would be use all the evaluations in one run of the GA with the largest population size possible. This may seem a plausible approach, because it is well known that the solution quality improves with larger populations. However, we could also use a smaller population and run the GA multiple times. Although the expected quality per run is expected to decrease, we would have more chances of reaching a good solution.

This paper examines the tradeoff between increasing the likelihood of success of individual runs vs. using more trials to reach the goal. The objective of this study is to determine what configuration reaches solutions with the highest quality. The scope of the study is limited to additively decomposable functions of bounded difficulty, because it is based on a model of solution quality that considers this type of functions.

It would be desirable to find that the best strategy is to use multiple small runs, because each could be executed concurrently on a different processor of a parallel computer resulting in a reduction of waiting time. Executing multiple isolated GAs in different processors is a bounding case of island-model parallel GAs, and has been studied in that context before (Tanese, 1989; Cantú-Paz & Goldberg, 1997). Tanese found experimentally that in some problems, the best overall solution found in any generation by multiple populations was at least as good as that found by a single run. Similarly, multiple populations showed an advantage when she compared the best individual in the final generation. However, when she compared the average population quality at the end of the experiments, the single runs seemed beneficial.

Other studies also suggest that multiple isolated runs can be advantageous. For example, Shonkwiler (1993) used a Markov chain model to argue that multiple small independent GAs would reach the global solution sooner (with fewer function evaluations) than a single GA with a large population. He suggested that superlinear parallel speedups are possible if the populations are executed concurrently on the nodes of a parallel computer.

Closer to our interests, Nakano, Davidor, and Yamada (1994) proved that under the constraint of constant computation cost there is an optimal population size that maximizes the probability of reaching a solution of a predetermined quality, and there is an optimal number of runs associated to this optimal size. Our approach is similar to theirs (although it has a different form), but we focus on certain types of functions and extend the analysis to identify the cases when multiple runs are beneficial based on certain characteristics of the problems that describe their difficulty.

More recently, Fuchs (1999) and Fernández et al. (2000) studied multiple isolated runs of genetic programming. They found that in some cases it is advantageous to use multiple small runs, but that it is important to balance the number of runs and their sizes to achieve good results.

Although there is some evidence that suggests that at least in some problems multiple runs are preferable, it is not entirely clear under what conditions this holds. For example, what does "better" mean? As we saw above, Tanese found conflicting answers depending on the comparison criterion.[1] Also, if multiple populations are indeed preferable, why are more practitioners not using them? We would expect that practitioners as "rational consumers" of ideas that have the potential to save time or reach better solutions would adopt multiple small runs. This paper will argue that the conditions where multiple runs are preferable are of limited practical value.

It is also interesting to consider the extreme cases when multiple runs of size one are better (in whatever sense) than a single GA, because multiple runs with a single individual are equivalent to random search. Although, it is known than in some problems, random search must be better than GAs (e.g., (Wolpert & Macready, 1997)) it is not clear on what problems this occurs. This paper sheds some light on this topic.

The models of this paper are based on the gambler's ruin (GR) model of Harik et al. (1999), which is summarized in the next section. Section 3 extends the GR model to calculate the expected quality of the best of r independent runs, and shows the circumstances under which it is advantageous to use multiple runs. Section 4 presents the results of experiments that validate the accuracy of the models. Section 5 relaxes the assumption of constant total cost and briefly discusses multiple short runs. Finally, section 6 presents a summary and the conclusions of this study.

2 THE GAMBLER'S RUIN MODEL

It is common in GA practice to encode the variables of the problem as binary strings. Although alphabets of higher cardinalities may be used, without loss of generality we restrict the discussion to the binary case. A schema is a string over the extended alphabet $\{0, 1, *\}$, and represents the class of individuals that have 0 or 1 in exactly the same positions as the schema. The * is a "don't care" symbol that matches anything. For example, in a domain that uses 10-bit strings, the class of individuals that start with 1 and have a 0 in the second position are represented by the schema 10********.

The number k of fixed positions in a schema is its order. The fixed positions of a schema define a partition of the search space into mutually exclusive subsets or equivalence classes. Some schemata represent classes of individuals with a higher average fitness than others, and some schemata actually match portions of the global solution. It is possible that a schema has a high average fitness, but does not match the global optimum. Low-order highly-fit schemata are sometimes called building blocks (BBs) (Goldberg, 1989). In this paper we refer to the lowest-order schema that consistently leads to the global optimum as the correct BB. In this view, the correct BB must (1) match the global optimum *and* (2) have the highest average fitness of all the schemata in the same partition. We label all other schemata in the partition as incorrect.

Harik et al. (1999) modeled selection in GAs as a biased random walk to obtain a model of the quality of the solution of a GA. Their work is based on a previous population sizing model by Goldberg, Deb, and Clark (1992). The model concentrates on only one partition of order k, and it assumes that decisions are independent across partitions. The number of copies of the correct BB in a population of size n is represented by the position, x, of a particle on a one-dimensional space, as depicted in figure 1. Absorbing barriers at $x = 0$ and $x = n$ bound the space, and represent ultimate convergence to the wrong and to the right solutions, respectively. Once the particle reaches the barriers it cannot escape. The initial position of the particle, x_0, is the number of copies of the correct BB in the initial population.

At each step of the random walk there is a probability, p, of obtaining one additional copy of the correct BB. This probability depends on the particular problem that the GA is facing, and it represents the probability of deciding correctly in a one-to-one competition between individuals that represent the best and the second best schemata of the partition. For functions composed by adding several uniformly-scaled subfunctions, p was computed by Goldberg, Deb, and Clark (1992) in their study of population sizing as

$$p = \Phi\left(\frac{d}{\sigma_{bb}\sqrt{2m'}}\right), \tag{1}$$

where Φ denotes the cumulative distribution function (CDF) of a normal distribution with a mean of zero and a standard distribution of one, d is the difference of the fitness contribution of the best and the second best schemata in the partition, $m' = m - 1$, m is the number of subfunctions, and σ_{bb}^2 is the average RMS variance of k-th order partitions.

A well-known result about random walks is the probability that a particle will eventually be captured by the absorbing barrier at $x = n$ (Feller, 1966):

$$P_{bb}(x_0, n) = \frac{1 - \left(\frac{q}{p}\right)^{x_0}}{1 - \left(\frac{q}{p}\right)^{n}} \tag{2}$$

[1] This paper compares algorithms based on the average fitness of solutions at the end of a run, and the results are consistent with Tanese's experiments.

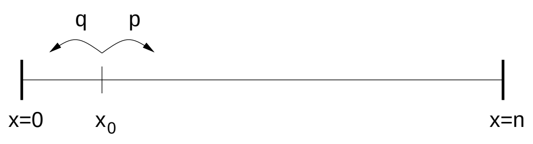


Figure 1: The bounded one-dimensional space of the gambler's ruin problem.

where $q = 1 - p$. We measure the quality of the solutions as the number of partitions that converged to the correct BB at the end of a run. Therefore, the expected probability of success is

$$P_s(n) = \sum_{x_0=0}^{n} P_0(x_0) \cdot P_{bb}(x_0, n), \tag{3}$$

where $P_0(x_0) = \binom{n}{x_0}\left(\frac{1}{2^k}\right)^{x_0}\left(1 - \frac{1}{2^k}\right)^{n-x_0}$ is the probability of having exactly x_0 correct BBs in the initial random population. In most cases, we may approximate the probability of success as $P_{bb}(\frac{n}{2^k}, n)$, which simplifies the calculations significantly, but in this paper we will use equation 3 to have more accurate results at small population sizes. Note that when $n = 1$ the approximation would return a value lower than the correct $\frac{1}{2^k}$.

There are a number of assumptions in the gambler's ruin model. First, there is no explicit notion of generations: the model considers that decisions in a GA occur one at a time until all the n individuals in its population converge to the same value. The model also assumes conservatively that all competitions occur between pairs of strings with the best and the second best schemata in a partition, and that the probability of deciding correctly remains constant during the run. Furthermore, the boundaries of the random walk are absorbing; this means that once a partition contains n copies of the correct BB it cannot lose one, and likewise, when the correct BB disappears from a partition there is no way of recovering it. This makes the implicit assumption that mutation and crossover do not create or destroy significant numbers of BBs: the only source of BBs is the random initialization of the population. Although, the GR model makes all these assumptions and simplifications, previous experimental results with additively decomposable functions of varying difficulty suggest that is it an accurate predictor of the solution quality (Harik et al., 1999). The models presented in the next section are largely based on the GR model, and therefore they inherit the assumptions, limitations, and applicability of the GR model.

3 MULTIPLE SMALL RUNS

Recall that we measure the quality of the solution as the number of partitions that converge correctly, and we denote it as Q. The probability that one partition converges correctly is given by the gambler's ruin model, $P_s(n)$. For convenience, we will use $P_1 = P_s(n_1)$ to denote the probability that a partition converges correctly in one run with population size n_1 and $P_r = P_s(n_r)$ for the probability that a partition converges correctly in one of the multiple runs with a population size n_r.

Under the assumption that the m partitions are independent, the quality has a binomial distribution with parameters m and P_r. Therefore, the expected solution quality of a single run is $\mathrm{E}(Q) = mP_r$. Of course, some runs will reach better solutions than others. If we execute r runs, we may write the qualities of the solutions that each reaches as

$$Q_{1:r} \leq Q_{2:r} \leq \ldots \leq Q_{r:r}.$$

These are the order statistics of the solution quality of the r runs. $Q_{1:r}$ denotes the quality of the worst solution, and $Q_{r:r}$ denotes the quality of the best solution found. We are interested in the expected value of $Q_{r:r}$, which can be calculated as (Arnold, Balakrishnan, & Nagaraja, 1992)

$$\mathrm{E}(Q_{r:r}) = \sum_{i=0}^{m-1} (1 - F(x)^r), \tag{4}$$

where $F(x) = P(Q \leq x) = \sum_{j=0}^{x} \binom{m}{j} P_r^j (1 - P_r)^{m-j}$ is the CDF of the solution quality. Unfortunately, there is no closed-form expression for the mean values of the maximal order statistics of binomial distributions, but their values can be found in tables or calculated numerically. However, there are approximations for the maximal order statistics of the standard normal distribution (Harter, 1970). To take advantage of this, we can approximate the binomial distribution of the quality with a Gaussian, and normalize the number of correct partitions by subtracting the mean and dividing over the standard deviation: $Z_{i:r} = \frac{Q_{i:r} - mP_r}{\sqrt{mP_r(1-P_r)}}$. The expected value of the highest normalized order statistic of r samples from a standard Gaussian is denoted as $\mathrm{E}(Z_{r:r}) = \mu_{r:r}$. Therefore, we can approximate the expected value of the best quality in r runs as

$$\mathrm{E}(Q_{r:r}) \approx mP_r + \mu_{r:r}\sqrt{mP_r(1 - P_r)}. \tag{5}$$

Noting that $\mu_{r:r}$ can be approximated accurately as $\mu_{r:r} \approx \sqrt{\sqrt{2}\ln r}$, we can see that the expected value of the best quality increases very slowly as more runs are used. Thus, if there were no restrictions on the total computations, adding more runs to an experiment would result in a marginally higher expected solution quality.

However, if the total cost is constrained, equation 5 shows a fundamental tradeoff: $\mu_{r:r}$ grows as r increases, but P_r decreases because the population size per run must decrease. This tradeoff suggests that there is an optimal number of runs and an associated optimal population size that maximize the expected quality. Unfortunately, even with the approximations we made we cannot obtain a closed-form expression for these optimal parameters, but with the model for the quality over multiple runs in place, we can start answering interesting questions. For example: Do multiple runs result in a higher quality than a single run that uses the same total computational resources? If so, what kinds of problems would benefit more?

The quality reached by multiple runs is better than one run if the following inequality holds:

$$mP_r + \mu_{r:r}\sigma_r > mP_1, \tag{6}$$

where $\sigma_r = \sqrt{mP_r(1-P_r)}$. We can bound the standard deviation as $\sigma_r = 0.5\sqrt{m}$ to obtain an upper bound on the quality of the multiple runs. Substituting this bound into the inequality above, dividing by m, and rearranging we obtain

$$\frac{\mu_{r:r}}{2\sqrt{m}} > P_1 - P_r. \tag{7}$$

This equation shows an interesting relation: multiple runs are more likely to be beneficial on short problems where m is small, everything else being equal. This is bad news for the case of multiple runs, because interesting problems in practice may be very long. The experiments in section 4 confirm that the advantage of multiple runs decreases in long problems.

The right side of the inequality above also shows that for multiple runs to be advantageous, the difference between the solution qualities must be small. This may happen at very small population sizes, where the quality is so poor that even in the single-run case it is very small. This case is not very interesting, because normally we want to find solutions with high quality. However, the difference is also small when the quality does not improve much after a certain population size. This is the case that Nakano, Davidor, and Yamada (1994) examined, and it opens an interesting possibility where multiple runs can be beneficial. The optimum population size is probably near the point where there is no further improvement. Using a larger population would be a waste of resources, and using multiple runs increases the chance of success.

3.1 MODELS OF CONVERGENCE TIME

We can write the total number of function evaluations that are available as

$$T = rgn_r, \tag{8}$$

where g is the domain-dependent number of generations until the population converges to a unique value, r is the number of runs, and n_r is the population size of each run.

In the remainder we assume that the generations until convergence are constant. Therefore, to maintain a fixed total cost, the population size of each of the multiple runs must be $n_r = n_1/r$, where n_1 denotes the population size that a single run would use.

Assuming that g is constant may be an oversimplification, as there is previous research that shows that the convergence time depends on other factors such as the population size and the selection intensity, I. For example, Mühlenbein and Schlierkamp-Voosen (1993) determined that under some conditions the generations until convergence are given by

$$g \approx \frac{\pi}{2}\frac{\sqrt{n}}{I}. \tag{9}$$

If we used this model, the population size of each of the multiple runs would have to be $n_r = n_1/r^{2/3}$ to keep the total cost constant. Using this form of n_r would give an advantage to the multiple runs, because their sizes (and the quality of their solutions) would not decrease as much as with the constant g assumption. In any case, the conclusions of the paper remain the same regardless of the convergence time model we use.

3.2 RANDOM SEARCH

Using all the available computation time in one run with the largest possible population is clearly one extreme in the spectrum of possibilities that we are considering. The other extreme is to use as many runs as possible with the smallest population size, which is one individual. Multiple runs with one individual are equivalent to random search, because there is no evolution possible (remember that we are assuming no mutation): the final result of each run is simply the random individual created at the beginning.

The models above account for the two extreme cases. When the population size is one, $P_r = \frac{1}{2^k}$, because only one term in equation 3 is different from zero. The quality of the best solution found by r runs of size one can be calculated with equation 4.[2]

To identify the problems where random search can find better solutions we calculated the expected solution quality using equation 4 varying the order of the BBs, k, and the number of runs. The next section will explain in more detail how the functions used in these calculations are defined; for now we only need to know that k varied. Figure 2 displays the results and shows the ratio of the quality obtained by random search over

[2]Taking $Q_{r:r} = m[1-(1-\frac{1}{2^k})^r]$ may seem tempting, but it greatly overestimates the true quality. This calculation implicitly assumes that the final solution is formed by correct BBs that may have been obtained in different runs.

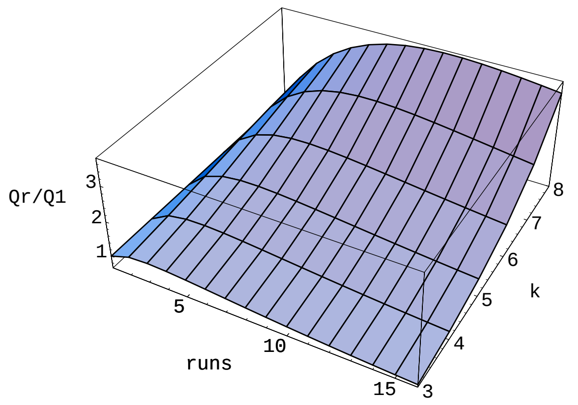


Figure 2: Ratio of the quality of multiple runs of size 1 (random search) vs. a single run varying the order of the BBs and the number of runs.

the quality found by a simple GA with a population size of $n_1 = r$. The figure shows that random search has a greater advantage as the order of the BBs increases; in some cases the quality it is more than three times higher than that found by the GA.

Taken literally, these results suggest that as the problems become harder (with longer BBs) random search is a better alternative than a GA. However, this peculiar behavior occurs only at extremely low population sizes, where the solution quality is so low that it is of no practical importance. When we increase the population size (and the number of random search trials), the GA comes ahead in the comparisons. Perhaps the main importance of these results is that they show that multiple runs can be beneficial in some cases, although not very practical ones.

A consequence of these results is that if the multiple random trials are executed in parallel and we calculate the parallel speedup using the simple GA as the base case, we would obtain superlinear speedups. Shonkwiler (1993) considered parallel execution and obtained superlinear speedups on several test functions. It is interesting to note that he used very small population sizes in each run (≈ 2 individuals), and at least two of his test functions are easily solvable by random search.

4 EXPERIMENTS

This section describes experiments with several linearly-decomposable functions of varying difficulty. The test functions were chosen to correspond to the ones used by Harik et al. (1999) in their study of population sizes, because they demonstrated that the GR model, which forms the basis of the models of this paper, accurately predicts the quality reached by a simple GA.

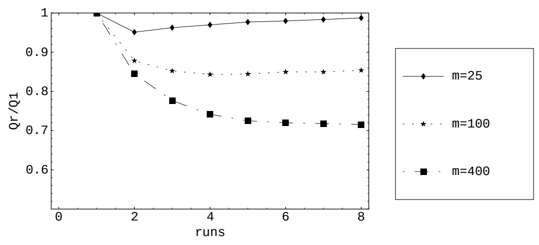


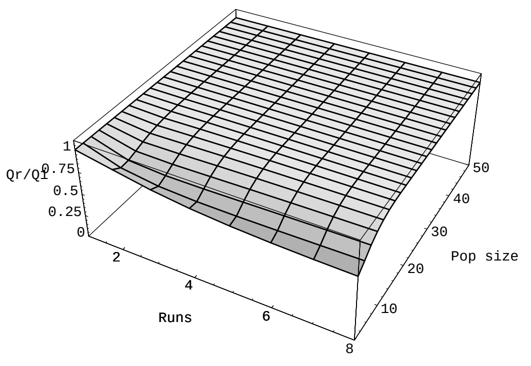
Figure 4: Ratio of the quality of multiple runs vs. a single run varying the problem size.

The GA in the experiments used pairwise tournament selection without replacement, one-point crossover with probability 1 (except where noted), and no mutation. All the results shown in this section are the average of 200 trials with each configuration.

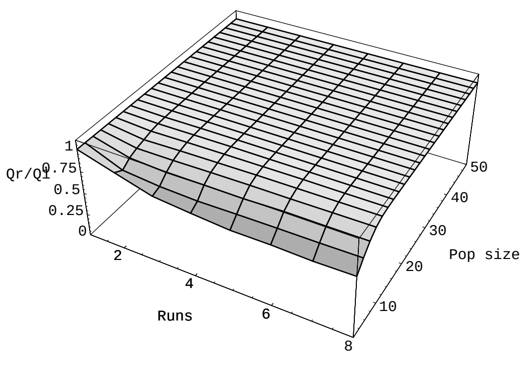
The first function is a simple one-max or counting ones function. It is defined as $f_1 = \sum_{i=1}^{m} x_i$, where $x_i \in \{0, 1\}$ are the individual bits of the chromosomes. This is an easy function with BBs of order $k = 1$. Our test function had a length of $m = 25$ bits, and p can be calculated to be 0.6135 using equation 1. In the first set of experiments, we varied the population size n_r from 2 to 50 individuals. For each population size, we varied the number of runs from 1 to 8 and recorded the quality of the best solution found in any of the runs, $Q_{r:r}$. Figure 3 shows the ratio of $Q_{r:r}$ over the quality Q_1 that a GA with a population size $n_1 = rn_r$ reached. The experiments match the predictions well, and in all cases the larger single runs reached solutions of better quality than the multiple smaller runs.

To illustrate that multiple runs are more beneficial when m is small, we conducted experiments varying the length of the problem. The additional experiments used the one-max function with $m = 100$ and $m = 400$ bits. The population size per run was fixed at $n_r = 10$, and the number of runs varied from 1 to 8. The empirical results plotted in figure 4 clearly show that as the problems become longer, the single large runs find better solutions than the multiple runs. Note that although $Q_{r:r}$ is increases with more runs, in general the ratio $Q_{r:r}/Q_1$ decreases.

The next two test functions are formed by adding fully-deceptive trap functions (Deb & Goldberg, 1993). As with the onemax, the values of the deceptive trap functions depend on the number of bits set to one in their k-bit input, u, but the fitness increases with more bits set to zero until it reaches a local optimum. The global maximum is at the opposite extreme where all the bits in the partition are set to one. The order-k traps are



(a) Theory



(b) Experiments

Figure 3: Ratio of the quality of multiple runs vs. a single run for the onemax test function with $m = 25$ bits.

defined as

$$f_{\text{dec}}^{(k)}(u) = \begin{cases} k - u - 1 & \text{if } u < k, \\ k & \text{if } u = k. \end{cases} \qquad (10)$$

The first deceptive test function is formed by concatenating $m = 25$ copies of $f_{\text{dec}}^{(3)}$. The fitness of a string is calculated as $\sum_{i=0}^{m-1} f_{\text{dec}}^{(3)}(u_{3i})$, where u_{3i} denotes the number of ones in the substring that starts at position $3i$. For this function, $p = 0.5573$. Similarly, the second deceptive test function is formed with $m = 25$ copies of $f_{\text{dec}}^{(4)}$: $\sum_{i=0}^{m-1} f_{\text{dec}}^{(4)}(u_{4i})$, and $p = 0.552$. Figures 5 and 6 show the ratio of the expected qualities in multiple over single runs, $Q_{r:r}/Q_1$, varying the run size from 2 to 100 individuals and the number of runs from one to eight. The experimental results are very close to the predictions, except with very small population sizes. The discrepancy is due to the inaccuracy of the GR model for very small population sizes. Observe that in most cases, the ratio is less than one, indicating that a single large run reaches a solution with better quality than multiple small runs. The exceptions occur at very small population sizes, where most of the time random search performs even better.

We performed experiments to validate the results about random search. Figure 7 shows the ratio of the quality of the solutions found by the best of r random trials and the solution obtained by a GA with a population size of r. For each value of k from 3 to 8, the test functions were formed by concatenating $m = 25$ order-k trap functions. The predictions in figure 2 correspond to the same test functions. The experiments show the same general tendency as the predictions.

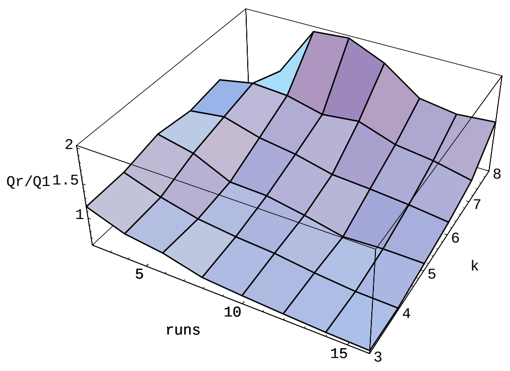
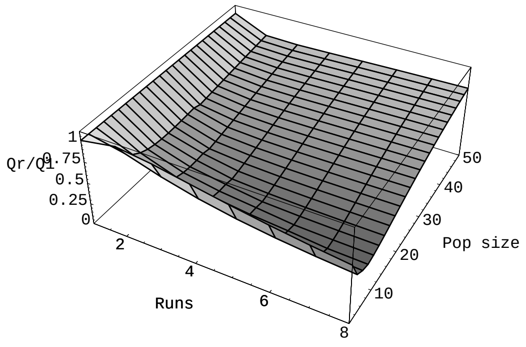


Figure 7: Ratio of the quality of multiple runs of size 1 (random search) vs. a single run varying the order of the BBs and the number of runs.

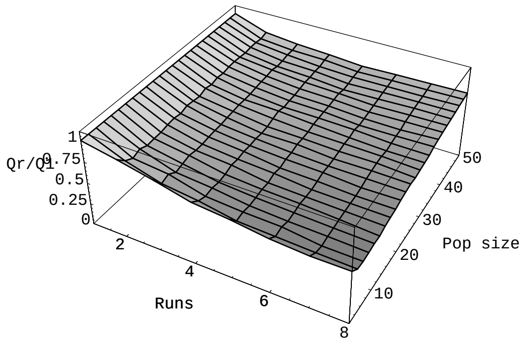
5 MULTIPLE SHORT RUNS

Until now we have examined the solution quality after the population converges to a unique solution, and no further improvement is possible. However, in practice it is common to stop a GA run as soon as it finds a solution that meets some quality criterion. The framework introduced in this paper could be applied to this type of experiments, if we had a model that predicted the solution quality as a function of time: $P_s(n, t)$. In any generation (or any other suitable time step), the expected solution quality in one run would be $mP_s(n, t)$, but again we would be interested in the expected value of the best of the r runs, which can be found by substituting the appropriate distribution in equation 4.

There are some existing models of quality as a function of time, but they make the assumption that the population size is sized such that the GA will reach

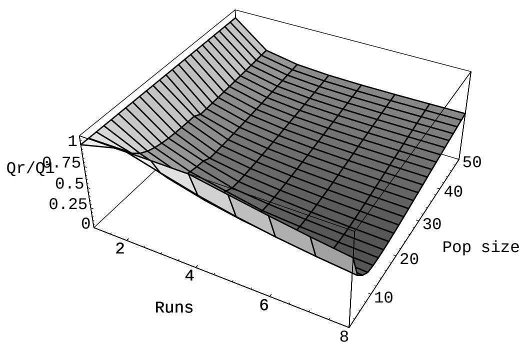


(a) Theory

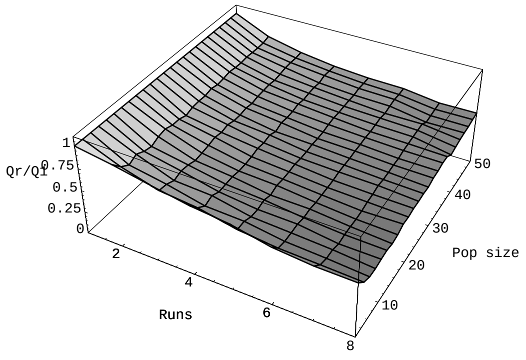


(b) Experiments

Figure 5: Ratio of the quality of multiple runs vs. a single run for the order-3 deceptive test function.



(a) Theory



(b) Experiments

Figure 6: Ratio of the quality of multiple runs vs. a single run for the order-4 deceptive test function.

the global solution and that recombination of BBs is perfect (Mühlenbein & Schlierkamp-Voosen, 1993). If we adopt these assumptions, we could use the existing models, but we would not be able to reduce the population size to respect the constraint of fixed cost.

Mühlenbein and Schlierkamp-Voosen (1993) derived the following expression for the one-max function:

$$P_s(n,t) = \frac{1}{2}\left(1 + \sin(\frac{I}{\sqrt{n}}t)\right), \qquad (11)$$

and Miller and Goldberg (1996) used it successfully to predict the quality of deceptive functions. If we abandon the cost constraint, we can study a different facet of the question of multiple vs. single runs. We can show that the best of multiple runs of the same size (that is at least large enough to reach the global optimum) reaches the solution faster than a single run of the same size. Figure 8 shows the ratio of the number of generations until convergence (to the global) of multiple runs over the number of generations of convergence of a single run. The figure shows that the time decreases as more runs are used, and the advantage is more pronounced for shorter problems. If each run is executed on a different processor of a parallel machine, the time to reach the solution would be reduced. However, note that this scheme offers a very small advantage, and it is probably not the best use of multiple processors since we can obtain almost linear speedups in other ways.

6 SUMMARY AND CONCLUSIONS

The dilemma of using one or multiple runs has interested researchers for a long time, but it has remained largely ignored despite multiple reports of the advantage of multiple runs. The paper considered ad-

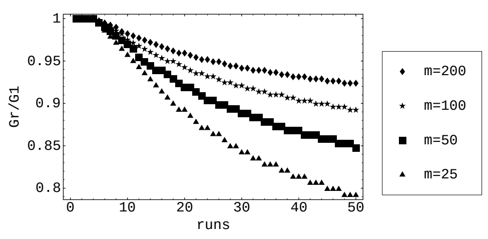


Figure 8: Ratio of the generations until convergence of multiple over single runs. The total cost is not constant.

ditively decomposable functions and extended an existing model to predict the solution quality based on the number of runs and the population size of each. Under the constraint of fixed total cost, this paper showed that the expected solution reached by multiple small runs is better than the solution reached by a single large run only in very limited conditions of no practical importance. In particular, multiple runs are advantageous when the difference between the qualities reached by single and multiple runs is small, which occurs when the solution qualities are both very small or when increasing the population size does not result in a better solution. In addition, the paper identified that the greatest advantage of multiple runs is on short problems. The results are consistent with previous experimental and theoretical studies.

There are ample opportunities for future research. Nakano et al. (1994) proved that under a constant cost constraint there is an optimal run size and count that maximize the chances of success. The models in this paper could be used to find this optimum numerically, but it would be interesting to characterize it analytically. Another extension is to consider different classes of fitness functions and other evolutionary algorithms. In addition, there is some existing experimental evidence that suggests that if we change the criterion used to compare algorithms the conclusions might change. Although the conclusions remained the same after our preliminary study comparing the quality of multiple runs over time, this case merits additional consideration.

The main contribution of this paper was to identify the problem characteristics (length and BB order) where it might be advantageous to use multiple runs. The results suggest that for difficult problems our best bet is to use a single run with the largest population possible. Small independent runs should be avoided.

References

Arnold, B., Balakrishnan, N., & Nagaraja, H. N. (1992). *A first course in order statistics*. New York, NY: John Wiley and Sons.

Cantú-Paz, E., & Goldberg, D. E. (1997). Modeling idealized bounding cases of parallel genetic algorithms. In Koza, J. R. et al. (Eds.), *Genetic Programming 97* (pp. 353–361). San Francisco, CA: Morgan Kaufmann Publishers.

Deb, K., & Goldberg, D. E. (1993). Analyzing deception in trap functions. In Whitley, L. D. (Ed.), *Foundations of Genetic Algorithms 2* (pp. 93–108). San Mateo, CA: Morgan Kaufmann.

Feller, W. (1966). *An introduction to probability theory and its applications* (2nd ed.), Volume 1. New York, NY: John Wiley and Sons.

Fernández, F., Tomassini, M., Punch, W., & Sánchez, J. M. (2000). Experimental study of isolated multipopulation genetic programming. In Whitley, D. et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 2000* (pp. 536). San Francisco, CA: Morgan Kaufmann Publishers.

Fuchs, M. (1999). Large populations are not always the best choice in genetic programming. In Banzhaf, W. et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference 1999: Volume 2* (pp. 1033–1038). San Francisco, CA: Morgan Kaufmann Publishers.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Harik, G., Cantú-Paz, E., Goldberg, D., & Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, *7*(3), 231–253.

Harter, H. L. (1970). *Order statistics and their use in testing and estimation*. Washington, D.C.: U.S. Government Printing Office.

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, *4*(2), 113–131.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Nakano, R., Davidor, Y., & Yamada, T. (1994). Optimal population size under constant computation cost. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving fron Nature, PPSN III* (pp. 130–138). Berlin: Springer-Verlag.

Shonkwiler, R. (1993). Parallel genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 199–205). San Mateo, CA: Morgan Kaufmann.

Tanese, R. (1989). Distributed genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 434–439). San Mateo, CA: Morgan Kaufmann.

Wolpert, D., & Macready, W. (1997). No-free-lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82.